Making a Century in HERMIT

Neil Sculthorpe, Andrew Farmer, and Andrew Gill

Functional Programming Group
Information and Telecommunication Technology Center
University of Kansas

2nd October 2014

KU

Compilers Should not be Black Boxes

- We improve spam filters by scripting.

- Can we fix our compiler using scripting?

Remote Shell for our Haskell compiler?

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- This idiom has many instantiations: faster code; using a different interface; space usage; semi-formal verification.
- We want to mechanise such transformations on Haskell programs:
  - less time-consuming and error prone than pen-and-paper reasoning
  - no need to modify the source file
- Several existing transformation systems for Haskell programs, e.g. HaRe, HERA, PATH, Ultra. They all operate on Haskell source code.
- **We take a different approach, and provide commands to transforming GHC Core, GHC's intermediate language.**

KU

Demonstration: Unrolling Fibonacci

As a first demonstration, let's transform the *fib* function by unrolling the recursive calls once.

$$
\begin{aligned}
&\mathit{fib} :: \mathit{Int} \to \mathit{Int} \\
&\mathit{fib}\ \mathsf{n} = \textbf{if}\ \mathsf{n} < 2 \\
&\qquad\qquad \textbf{then}\ 1 \\
&\qquad\qquad \textbf{else}\ \mathit{fib}\ (\mathsf{n} - 1) + \mathit{fib}\ (\mathsf{n} - 2)
\end{aligned}
$$

KU

Demonstration: Unrolling Fibonacci

As a first demonstration, let's transform the *fib* function by unrolling the recursive calls once.

```
fib :: Int → Int
fib n = if n < 2
        then 1
        else fib (n − 1) + fib (n − 2)
```

```
fib :: Int → Int
fib n = if n < 2 then 1
                 else (if (n − 1) < 2 then 1
                                      else fib (n − 1 − 1) + fib (n − 1 − 2)
                      )
                      +
                      (if (n − 2) < 2 then 1
                                      else fib (n − 2 − 1) + fib (n − 2 − 2)
                      )
```

First Demo

First Demo

- resume ..... resume the compile
- `binding-of 'main` ..... goto the `main` definition
- `binding-of 'fib` ..... goto the `fib` definition
- `remember "myfib"` ..... remember a definition
- `show-remembered` ..... show what has been remembered
- any-call (unfold-remembered "myfib") ..... try unfold "myfib"
- `bash` ..... bash a syntax tree with simple rewrites
- `top` ..... go back to the top of the syntax tree
- `load-and-run "Fib.hss"` ..... load and run a script

KU

What did we do?

HERMIT requires a recent ghc (I am using GHC 7.8.3)

1. `cabal update`
2. `cabal install hermit`
3. `hermit Main.hs`

The `hermit` command just invokes GHC with some default flags:

```
% hermit Main.hs
ghc Main.hs -fforce-recomp -O2 -dcore-lint
            -fexpose-all-unfoldings
            -fsimple-list-literals -fplugin=HERMIT
            -fplugin-opt=HERMIT:main:Main:
```

KU

HERMIT Use Cases

- We want to explore the use of the worker/wrapper transformation for program refinement
  - We need mechanization to be able to scale the idea to larger examples
  - Medium-sized case study: Changing representations in the Conway's Game of Life
  - Are working on large case study: Low Density Parity Checker (LDPC), tansforming math equations into Kansas Lava programs
- HERMIT is for library writers
  - Authors show equivalence between clear (specification) code, and efficient (exported) code.
- HERMIT is a vehicle for prototyping GHC passes
  - Optimization: Stream Fusion
  - Optimization: SYB
  - Staging: Translating Core into CCC combinators. (Elliott, et. al.)
- Scripting Haskell-based Equational Reasoning – **this talk**
- (Your project goes here)

KU

Highest Level Architecture

We draw inspiration from UNIX and operating systems.

Three levels

- Shell Level ..... (UNIX Shell style commands)
- Rewrite Level ..... (UNIX man(2) system commands)
- Stratego-style library for rewrites ..... (DSL for rewrites)

KU

Shell Level

UNIX Shell style commands

- Dynamically typed, variable arguments
- Help (man) for each command
- Control flow commands (';', retry, etc.)

KU

Rewrite Level

UNIX man(2) system commands

- Haskell functions, strongly typed
- Think type :: `CoreExpr` $\rightarrow$ `M CoreExpr`
- Higher-order functions for tunneling into expressions
- Many function tunnel into GHC (example: substExpr)
- Allow, all GHC "RULES" are directly invokable.

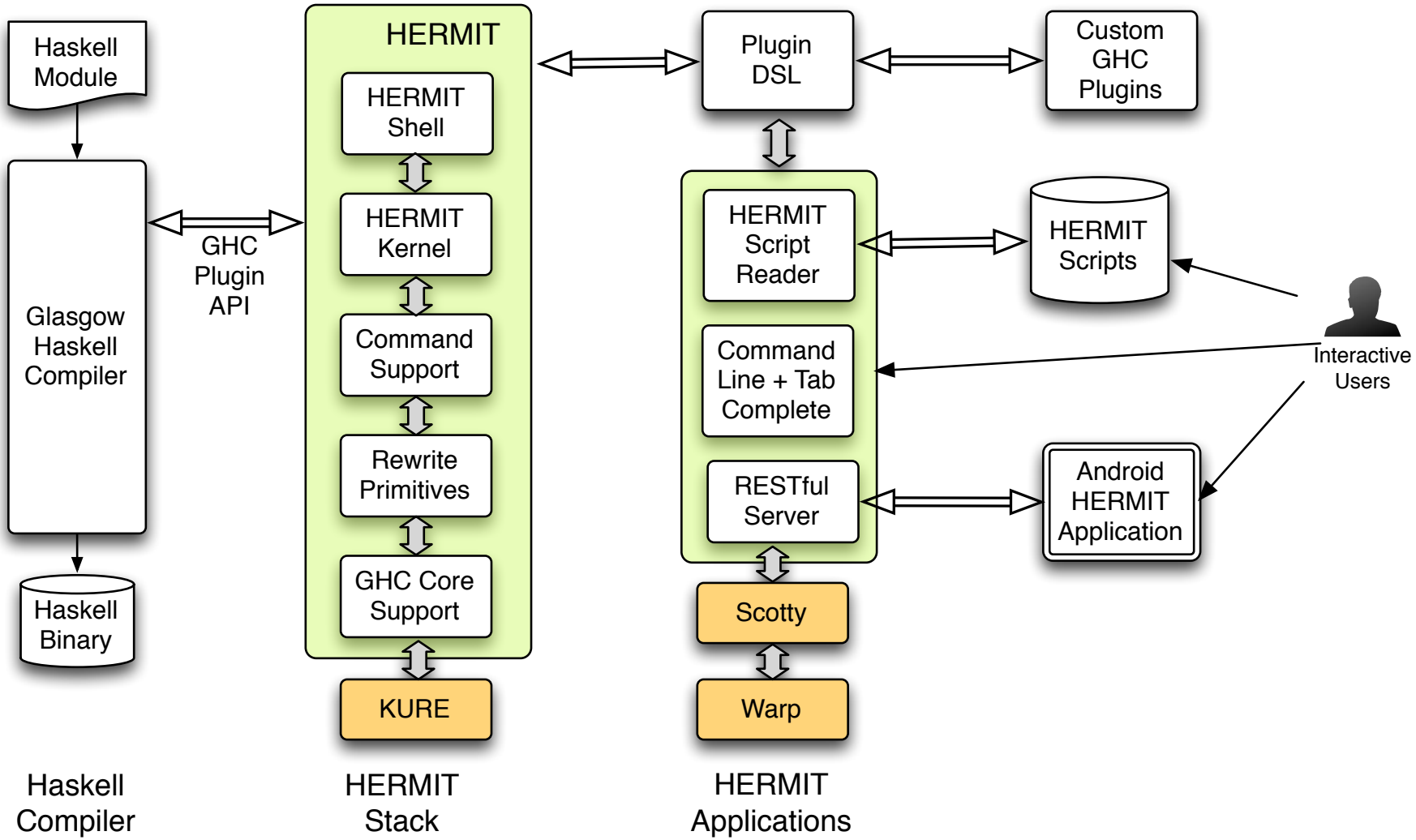
KU

Stratego style library for rewrites

- Haskell DSL call KURE
- Basic idea: rewrites can succeed or fail
- Higher-order combinators for search, catching fail, retry
- Both levels reflect the Stratego API

KU

Lifting the Lid on the HERMIT Project

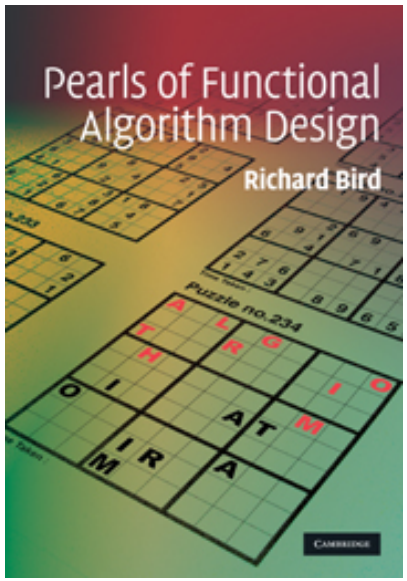


HERMIT Commands

- Core-specific rewrites, e.g.
  - beta-reduce
  - eta-expand *'x*
  - case-split *'x*
  - inline
- Strategic traversal combinators (from KURE), e.g.
  - any-td *r*
  - repeat *r*
  - innermost *r*
- Navigation, e.g.
  - up, down, left, right, top
  - binding-of *'foo*
  - app-fun, app-arg, let-body, ...
- Version control, e.g.
  - log
  - back
  - step
  - save "myscript.hss"

KU

Pearls of Functional Algorithm Design



- The book is entirely dedicated to reasoning about Haskell programs, with each chapter calculating an efficient program from an inefficient specification program.
- We selected the chapter *Making a Century* from the textbook *Pearls of Functional Algorithm Design*.

KU

Larger Example: Deriving a better century

The program in *Making a Century* computes the list of all arithmetic expressions formed from ascending digits, where juxtaposition, addition, and multiplication evaluate to 100. For example, one possible solution is

$$100 = 12 + 34 + 5 \times 6 + 7 + 8 + 9$$

The derivation of an efficient program involves a substantial amount of equational reasoning, and the use of a variety of proof techniques, including fold/unfold transformation, structural induction, fold fusion, and numerous auxiliary lemmas.

KU

Fragment of the proof:

```
{ rhs-of 'solutions

  -- filter (good . value) . expressions

        { app-arg ; inline 'expressions }

  -- filter (good . value) . foldr extend []

        { [ app-fun, app-arg ] ; apply-rule "6.2" }
        { lemma "comp-assoc" }

  -- filter (good . value) . filter (ok . value) . foldr extend []

        { app-arg

          -- filter (ok . value) . foldr extend []

              { lemma "foldr-fusion-1" }

          -- foldr extend' []

        }

  -- filter (good . value) . foldr extend' []

}
```

GHC RULES

- GHC language feature allowing custom optimisations
- e.g.

{-# RULES "map/map" $\forall$ f g xs. *map* f (*map* g xs) = *map* (f $\circ$ g) xs #-}

- HERMIT adds any RULES to its available transformations
  - allows the HERMIT user to introduce new transformations
  - HERMIT can be used to test/debug RULES
  - We use the `[~]` syntax to record, but not use, a rule.

KU

What happened while deriving a better century

- During mechanization we discovered that several auxiliary properties in the textbook are stated as assumptions without proof.
  - we suspect that they are deemed either "obvious" or "uninteresting".
- Assumption 6.2 also had a simple proof, but it relied on arithmetic properties of Haskell's built-in `Int` type (specifically, that `m == n` $\implies$ `m <= n`).
- Two proof techniques are used in the textbook that HERMIT does not directly support.
  - The first is the fold fusion law, which needs implication, which we do not support.
  - The second involves postulating the existence of an auxiliary function.
  - We did manage to run the postulated function backwards, to verify the calculation.
- We have a plugin that provides the fold fusion law as a primitive.

KU

Length of Calculations for Century

Calculation	Textbook Lines	HERMIT Commands		
		Transformation	Navigation	Total
solutions	16	12	7	19
expand	19	18	20	38
Lemma 6.5	not given	4	4	8
Lemma 6.6	not given	2	1	3
Lemma 6.7	not given	2	0	2
Lemma 6.8	7	5	8	13
Lemma 6.9	1	4	4	8
Lemma 6.10	not given	23	13	36
Total	43	70	57	127

KU

HERMIT Summary

- A GHC plugin for interactive transformation of GHC Core programs
- Can express basic equational reasoning as HERMIT scripts
- New and powerful commands can be defined using a HERMIT plugin mechanism

```
cabal install hermit
```

KU